same

Certainly! Here's a **clear, step-by-step explanation** of the code in your `dfs_recursive.py` file, along with how its output is produced.

What Does This Code Do?

- Lets you build an **undirected graph** by entering vertices and their neighbors.
- Asks for a **start** and **goal** vertex.
- Uses **recursive Depth-First Search (DFS)** to:
  1. **Traverse** the graph and print each vertex visited.
  2. **Find and print a path** from the start to the goal vertex.

Code Explanation (Line by Line)

Imports

```
from collections import defaultdict
```

- Imports `defaultdict` to easily create a graph where each vertex has a set of neighbors.

Recursive DFS Traversal

```
def dfs_recursive(graph, vertex, visited=None):
    if visited is None:
        visited = set()
    visited.add(vertex)
    print(vertex)  # Do something with the vertex
    for neighbor in graph[vertex]:
        if neighbor not in visited:
            dfs_recursive(graph, neighbor, visited)
```

- **Purpose:** Visits all vertices reachable from `vertex` using DFS.
- **visited** keeps track of already visited nodes to avoid loops.
- **For each neighbor**, if not visited, calls itself recursively to go deeper.
- **Prints each vertex** as it is visited.

Recursive DFS Path Finder

```
def dfs_path_recursive(graph, current, goal, path=None, visited=None):
    if visited is None:
        visited = set()
    if path is None:
        path = [current]
    visited.add(current)
    if current == goal:
        return path
    for neighbor in graph[current]:
        if neighbor not in visited:
            result_path = dfs_path_recursive(graph, neighbor, goal, path + [neighbor], vi
            if result_path:  # If path found in recursion
                return result_path
    return None  # No path found
```

- **Purpose:** Finds a path from `current` to `goal` using DFS.
- **visited**: tracks visited nodes.
- **path**: list of nodes from start to current.
- **If current is goal**, returns the path.
- **For each neighbor** not visited, recurses deeper, adding neighbor to path.
- **If a path is found** in recursion, returns it; otherwise, returns `None`.

Graph Input

```
graph = defaultdict(set)
num_vertices = int(input("Enter the number of vertices: "))
for _ in range(num_vertices):
    vertex = input("Enter a vertex: ")
    neighbors = input("Enter its neighbors (separated by spaces): ").split()
    graph[vertex].update(neighbors)
    for neighbor in neighbors:
        graph[neighbor].add(vertex)  # Ensure undirected connection
```

- **Builds the undirected graph** by taking input for each vertex and its neighbors.
- **Ensures** if `a` is a neighbor of `b`, then `b` is also a neighbor of `a`.

Start and Goal Input

```
start_vertex = input("Enter the starting vertex: ")
goal_vertex = input("Enter the goal vertex: ")
```

- Takes the start and goal vertices as input.

Running DFS Traversal

```
print("\nRecursive DFS traversal:")
dfs_recursive(graph, start_vertex)
```

- Prints a heading.
- Calls the recursive DFS traversal function, printing each visited vertex.

Running DFS Path Finder

```
print("\nRecursive DFS path to goal:")
path = dfs_path_recursive(graph, start_vertex, goal_vertex)
if path:
    print("Path to goal:", path)
else:
    print("Goal not found")
```

- Prints a heading.
- Calls the recursive DFS path function.
- If a path is found, prints it; otherwise, prints "Goal not found".

Sample Input and Output Explained

Input

```
Enter the number of vertices: 5
Enter a vertex: a
Enter its neighbors (separated by spaces): b c e
Enter a vertex: b
Enter its neighbors (separated by spaces): a d
Enter a vertex: c
Enter its neighbors (separated by spaces): a d
Enter a vertex: d
Enter its neighbors (separated by spaces): a c d
Enter a vertex: e
Enter its neighbors (separated by spaces): a d
Enter the starting vertex: a
Enter the goal vertex: c
```

Output

```
Recursive DFS traversal:
a
b
d
c
```

```
e

Recursive DFS path to goal:
Path to goal: ['a', 'b', 'd', 'c']
```

What's Happening?

1. **Graph is built** as per your inputs.
   - Example: `a` is connected to `b`, `c`, `e`, etc.
2. **DFS traversal** starts at `a`:
   - Visits `a`, then `b` (neighbor of `a`), then `d` (neighbor of `b`), then `c` (neighbor of `d`), then `e` (neighbor of `a`).
   - The order depends on the order of neighbors in the set.
   - Each visited vertex is printed as it is visited.
3. **DFS path search** from `a` to `c`:
   - Tries `a → b → d → c`.
   - Returns the path as soon as `c` is found.
   - Prints: `Path to goal: ['a', 'b', 'd', 'c']`.

Summary Table

Step	What it Does	Output Example
Build graph	User inputs vertices and neighbors	(No output)
DFS traversal	Prints each vertex as it is visited recursively	a, b, d, c, e
DFS path to goal	Prints one path from start to goal (if exists)	['a', 'b', 'd', 'c']

Key Points

- **DFS** explores as far as possible along each branch before backtracking.
- **Traversal** prints all reachable nodes from the start.
- **Path search** returns one path from start to goal (not always the shortest).
- **If no path exists**, prints "Goal not found".

If you want to see how the code would behave with different inputs or want a visual diagram, just ask!

⁂